Creating a Pose-Estimation System for Use in Studying Motor Cortical Dynamics

Valentine Wilson
Undergraduate Thesis
Spring 2020

Introduction

In computational neuroscience, a main focus in recent years has been on connecting human brain signals to devices to essentially allow "mind control" of machines. This area, dubbed brain-computer interfacing (BCI), explores technologies in many spaces including prosthetics and enhanced wearables. In addition to aiding the development of new technologies, being able to process and make sense of neural dynamics also helps us understand the mechanisms behind neurodegenerative diseases affecting the motor cortical system such as amyotrophic lateral sclerosis (ALS) and Parkinson's Disease. [1]

Currently, many researchers are looking to better understand motor control by looking at stereotyped tasks. Some researchers use lesion studies to try to uncover how the motor cortex is used in the execution of stereotyped tasks.[2] Other labs use variety of tools such as EMG[3], MEG[4], and EEG[5] to help understand the motor cortical system.

Our lab, specifically, uses a rodent paradigm where rats perform a supination task while simultaneously having population-level neuronal spiking activity recorded by a tetrode, which is a device employing four electrodes to record and triangulate neuronal spikes. Using this method, researchers can correlate the rat movements (the velocity of the turn, the angle the knob is turned to, the force needed to complete the turn, and the overall time spent turning) with activity occurring in the motor cortex. [6]

While these results are fascinating and provide interesting mappings between behavior and neuronal dynamics, they are quite limited in scope. Using this method, one can infer dynamics from knob turning tasks but not from more naturalistic tasks such as walking around the cage, eating, or cleaning fur. Furthermore, using readings from knobs to try to understand and correlate movement is indirect and leaves room for error in recording.[6]

I introduce an application of the transfer learning pose-estimation software DeepLabCut to solve this issue of recording the movements of rats during periods other than knob-turning. By using a deep learning classifier to tag individual body parts of the rat during each frame of video, I will

be able to measure the movement of the rat without stereotyped tasks such as knob-turning. This will give us a richer, more complete picture of how movements other than the basic knob-turning paradigm correlate with motor cortical neuronal dynamics and cut months of training time from future experiments. [7]

Literature Review

In the domain of motion capture, many new and exciting techniques have been developed in the past couple of years as computer vision has become a hot topic in research, but before this, labs relied on more rudimentary methods such as reflective markers.

The most popular technique in the past has been to attach reflective markers to the rodent's limbs. Then in post-processing, the marker's locations are isolated based on pixel intensity. While this method provides high accuracy tracking of specifically marked body parts it does not allow for flexibility. All parts that are to be tracked must be labeled with a marker prior to video capture. This means if researchers want to track another location after taking the video, they will need to attach another marker and record another video. The marker method has also proved to be distracting to the rats causing them to fidget and try to remove them which impacts the accuracy of the data.[8]

For a more precise description of rat movement, researchers use high-speed cameras to record rat behavior along with post-processing. [9] One downside of this method is that it requires the use of time-intensive post-processing algorithms. In order to run these algorithms researchers must create training data, annotate data, train models, evaluate models and the adjust learning parameters.

One such algorithm, OpenPose, uses a bottom-up approach. First, it detects all the body parts in the image then it attempts to associate them to particular subjects. It produces a heatmap of the likelihood of a body part being at a given location (confidence map). It then produces an affinity map of vectors which encodes associations between key points (joint location). The affinity map includes a 2D vector for each limb which encodes the direction the limb points along with its

location in the image. It uses these two maps along with a parsing algorithm to create a skeleton for a tagged figure. The parsing problem is framed as a maximum-weight bipartite graph-matching with body part detection candidates represented as nodes and all possible connections between candidates represented as edges. The weight of an edge is calculated using body part orientations to produce a confidence measure of how likely the two parts are connected.[10]

DeeperCut, tackles the problem of multi-human pose estimation using a deep residual neural network (ResNet) in combination with pairwise terms. Pairwise terms are calculated using estimates provided from tagged body parts to predict where the rest of the pose is in relation to them. [11]

DeepLabCut (DLC) borrows the pre-trained ResNet layer from DeeperCut, focusing only on the feature detectors used in the model and excluding the pairwise term calculations, which are more useful in multi-subject classification instances. The ResNet is trained on thousands of images from ImageNet, a popular image classification benchmark. Then instead of using ResNet's classification layer, it is removed and a deconvolutional layer is added, which gives a readout of spatial probability densities for each body part (likelihood a body part is at a specific location). During training, the weights in the network are adjusted such that the areas with labeled body parts are assigned high probability densities such that the model is able to learn feature detectors for specific body parts. I opted to use this algorithm as it uses transfer learning to make the classifier more efficient than the other two algorithms mentioned. Compared to classic post-estimation algorithms, it is able to achieve comparable accuracy with less training data. [7]

Methods and Materials

We decided to use a desktop computer to run the DeepLabCut system. The package comes with a docker file to aid installation. The docker file was incompatible with CentOS, the operating system currently being run, so I booted Ubuntu from a USB. In order to transmit data from the cameras to the desktop, I also upgraded the machine with USB3.1 ports to allow for quick data-transfer (10Gbit/s).

I trained a model using the DLC demo data, which took about a day to complete. I determined that this was too slow for our usage. I then established a pipeline for GPU-accelerated code using a GPU located in the lab which fulfilled DLC's requirements (8GB memory, Tensorflow 1.0+ with GPU support for Python 3 and CUDA).

To use DeepLabCut on our data, it was necessary to set up a camera system which can take high speed video of the rodent paradigm (Fig 1). The data needed for the algorithm must be from a high-speed camera in order to capture micro-movements in the rats. In the original DeepLabCut paper, the authors mention using cameras recording at a range of 100-340fps. Using this information, I selected two BlackflyS U3-16S2M-CS which records in monochrome at 226 fps.

The rodent paradigms are located in a shelf space. After taking measurements of the enclosure along with the shelf length, I determined it would not be possible to fit a camera which would record at a sufficient angle given the wires obscuring the camera's view. I setup the cameras on tripods, 6 inches from the knob task and 6 inches above the table. After deciding that taping down the cameras is not sufficient to ensure stability during recording, I looked into purchasing an optical breadboard. I measured the box to be 6.5x13.75 inches. The cameras rest 6 inches from the knob, along the narrow side. Factoring this in, I figured we would need a 18x18 inch breadboard. In addition, I ordered an arm, thread, and post to fix the camera to the breadboard at a 6-inch height.

Given the camera would need to rest close to the rat box to record the movement of the paws without exterior noise, I was able to calculate that the camera must rest 6 inches from the subject. I chatted with a representative at Edmund Optics about focal length. Given the camera pixel size is 3.45 μm, sensor size is 1/2.9 inches, horizontal field of view is 50.8mm, working distance is 152.4mm, and smallest feature is about ½ inchx1/2inch, we decided on a lens with a focal length of 12.5mm. The lens we chose, the 12.5mm M12 Imaging Lens from Edmund Optics, allows for adjustment to the working distance via threading, to correct for any error in the working distance calculation.

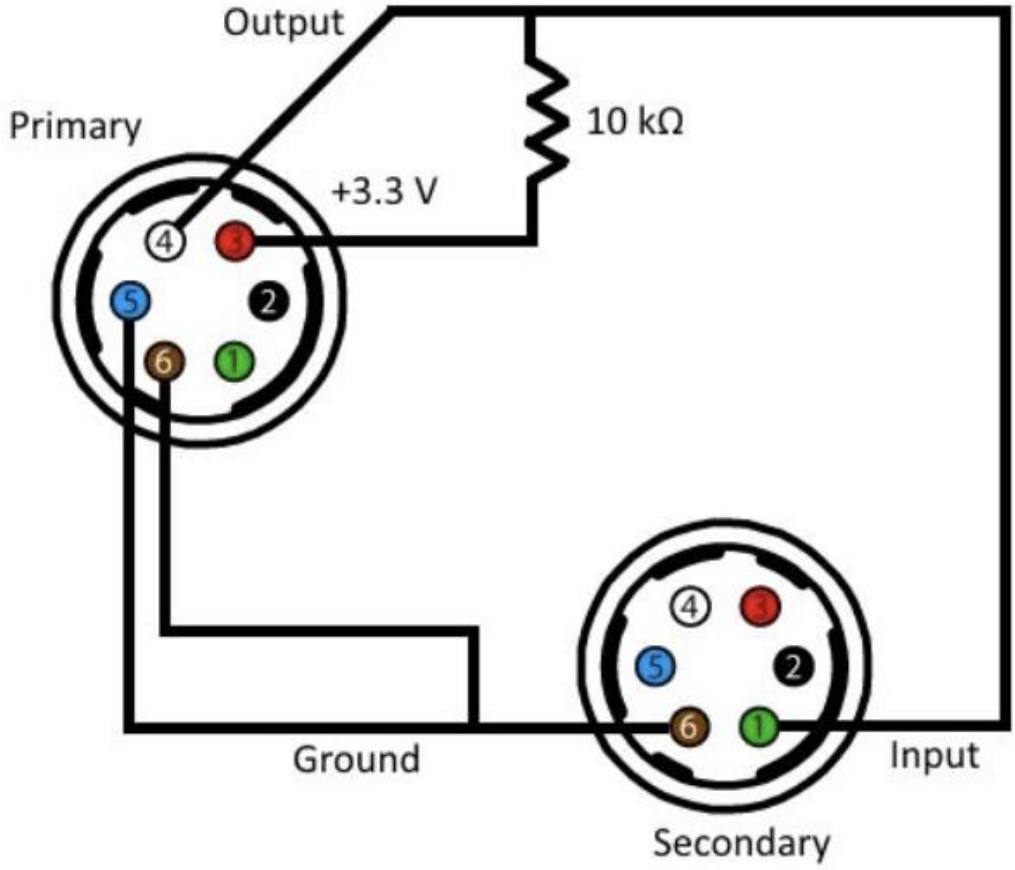

Fig 1. Wiring diagram for the primary/secondary device connection

In order to capture video from the two cameras at the same time, I had to physically connect the two cameras. This was done by soldering together the output channel and power channel from the primary camera's GPIO to the input channel of the secondary camera. I then linked their ground channels (Fig.1, Fig.2). According to this setup, when the primary camera is triggered it sends a strobe to the secondary camera causing it to trigger and capture an image.[12] I triggered the primary camera using SpinView, a program supplied by FLIR. To use the desktop machine as a trigger I connected it to the primary camera via the USB3.1 port I installed.

Green	1	3	V_{AUX}	Auxiliary Input Voltage (DC)
			GPI	Non-isolated Input
Black	2	0	OPTOIN	Opto-isolated Input
Red	3	2	VOUT	Camera Power Output
			GPIO	Non-isolated Input/Ouput
White	4	1	OPTOOUT	Opto-isolated Output
Blue	5	N/A	Opto GND	Opto-isolated Ground
Brown	6	N/A	GND	Camera Power Ground

Fig 2. Wire identification table

This chain is then attached to a Raspberry Pi Model B's GPIO ports (same setup as in Fig.2) such that it triggers the primary camera followed by the secondary camera. After successfully capturing images using a strobe from the Raspberry Pi, I modified the input stream to include timestamp information so the data could be synced with the tetrode recordings. The Raspberry Pi runs code which records neural data from the rats via tetrode. By recording timestamp data, it is able to sync the sampling of the neural and image data (about a sample every 4 ms) so that they can be used to correlate the rat's movements with neuronal spiking data in the future.

Once the video is recorded, it is analyzed using a model I created with the DeepLabCut2 package. In order to train the model, I annotated 200 frames of a rat completing a turning task. I used two GUI's to annotate the images. The first GUI, PoseLabGUI, was created by Kanishk Jain a graduate student at Berman Lab to aid in the DeepLabCut annotation process. The second GUI was as an addition in DeepLabCut2 fed the annotations directly into the model being trained, thus simplifying the annotation process.

Results

Preliminary DeepLabCut Trial

To test whether the cameras are taking video at a frame rate high enough to detect subtle motions in rat movement, I first tested one camera on the camera setup seen in Figure 3. In this setup, the camera's frame is half covered by a mirror reflecting my finger, such that I have a view of the same finger from multiple angles. I then labeled the knuckle on my middle finger and the tip of my pointer finger both in the mirrored image and in the actual frame.

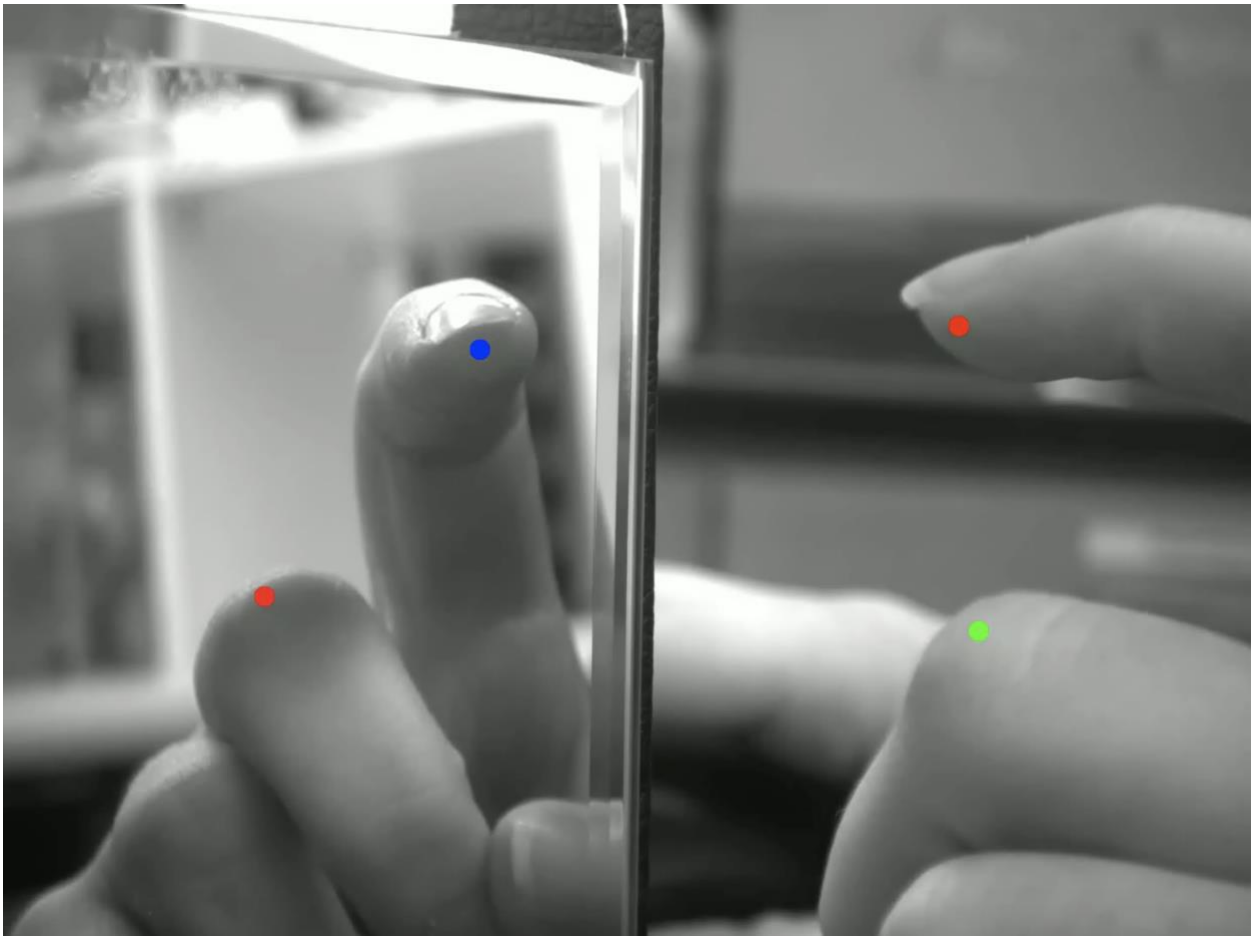

Fig 3. Example of a frame automatically labeled by the model

I produced training data by hand-annotating 200 of these images. I then trained a model using DeepLabCut with a 50-layer Resnet and a 90-10 training validation split. The resulting model was able to converge after about 20,000 iterations and has a test error of 2.1 pixels. I then tested the model on a 3-minute 11-second video of my finger wiggling. The trajectories of each finger are seen in Figure 4. You may notice the correlation between the movements of Second Knuckle and Second Knuckle 2 as well as between Pointer and Pointer 2. The model produces not only a map of finger trajectories but also a pickled CSV containing a column for each label along with its point in each frame. This CSV will be useful for finding the corresponding 3D points for each label.

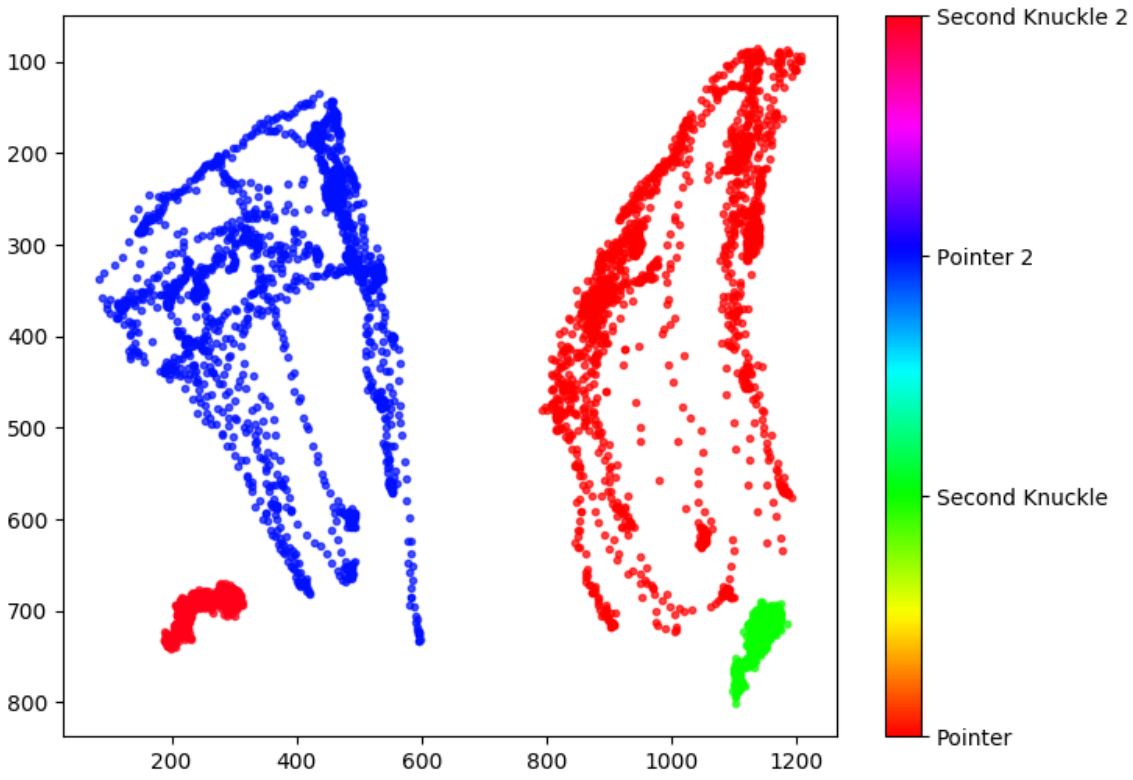

Fig 4. Finger trajectories, x/y-axes correspond to x/y pixel position on each frame. Taken from a 1-minute video corresponding to the frame in Figure 3.

3D Projection

To track rat movements in 3D, 2D points with the same label need to be merged. After communication with the DLC project team, I determined that the labels would need to be merged after DeepLabCut had been run as the algorithm is unable to label two views of the same point in the same frame (as in the finger-mirror example).

I looked into how to merge two 2D points into a 3D representation. The most popular method orthogonalizes the angle between the two points and projects the points into a 3D space using the pinhole-camera technique. In order to calculate the projected points, one needs the intrinsic and

extrinsic matrices to find the camera matrix.[13] One can decompose the camera matrix into the intrinsic and extrinsic matrices as such:

$P = K[R|t]$

Where P is represents the camera matrix, K represents the intrinsic parameters and [R|t] represents the extrinsic parameters. The matrix form of the equation is noted below[13]:

$$\mathbf{P} = \underbrace{\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsic parameters}} \underbrace{\left[\begin{array}{ccc|c} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{array}\right]}_{\text{extrinsic parameters}}$$

The intrinsic matrix details the distortion of each camera. This is represented by a 3x3 matrix with the p values describing the principal point and the f values representing focal length.[13]

The extrinsic matrix details the distance and angle between the two cameras and is represented by a 3x3 rotation matrix and a 3x1 translation vector [13]:

$$\underbrace{\mathbf{R} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}}_{\text{3D rotation}} \quad \underbrace{\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}}_{\text{3D translation}}$$

In order to find the intrinsic matrix for each camera, the Aruco chessboard method was used. This method required the use of the GitHub repository Camera-Calibration, along with 50 images of the chessboard taken by each camera.[14] I was able to validate this method by comparing the focal length of the lens to the focal length calculated by the calculation of the intrinsic matrices.

To find the extrinsic matrices, the cameras' positions need to be permanent or the matrices will need to be recalculated each trial. To fix the cameras in place, I set up an optical breadboard which the rat box and cameras can be mounted on.

Camera Angle Trials

Once I placed the cameras on the breadboard, I had to determine which angle would produce the highest quality videos for annotation. I conducted a series of trials to find the optimal angle.

During the first trial, I was unable to capture the details of the task due to visual noise from the enclosure. In Figure 5 you can see that the rat's paw is obscured by light bouncing off the enclosure while in Figure 6 you can see that the enclosure itself is blurring the image. I was able to remedy the angle in Figure 7 such that there is a clear view of the rat's hand. I did this by moving the camera into a clear view of the hand and turning the lights off. The second camera's position is still undetermined, but this angle will give me preliminary training data to at least begin annotating.

Fig 5. Camera frame obstructed by the enclosure

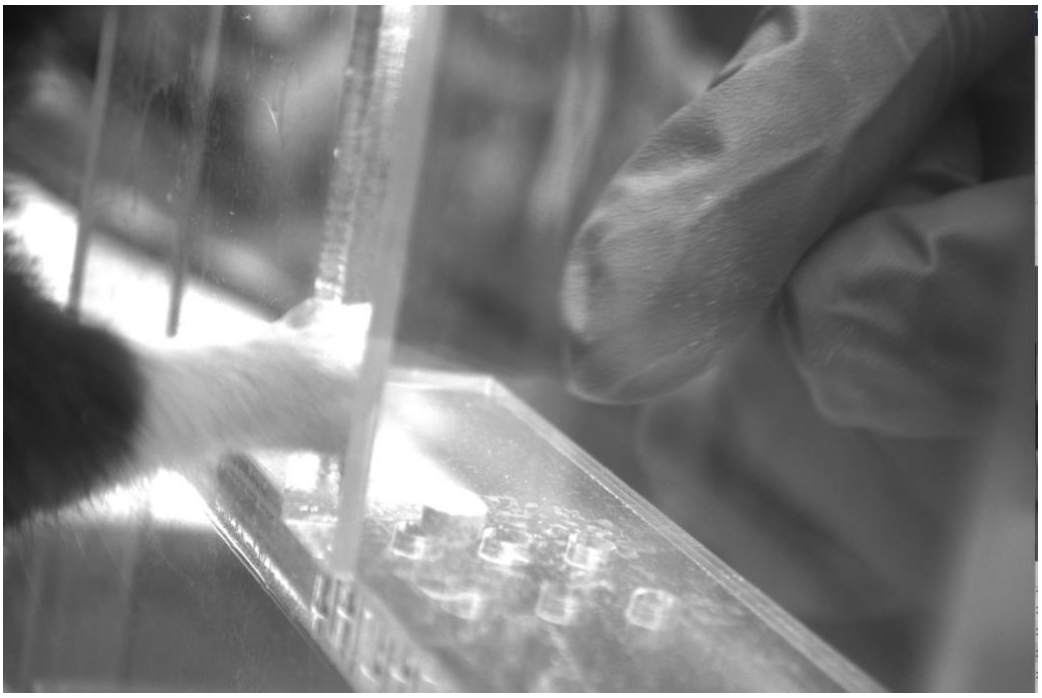

Fig 6. Rat hand obstructed by light bouncing off the enclosure

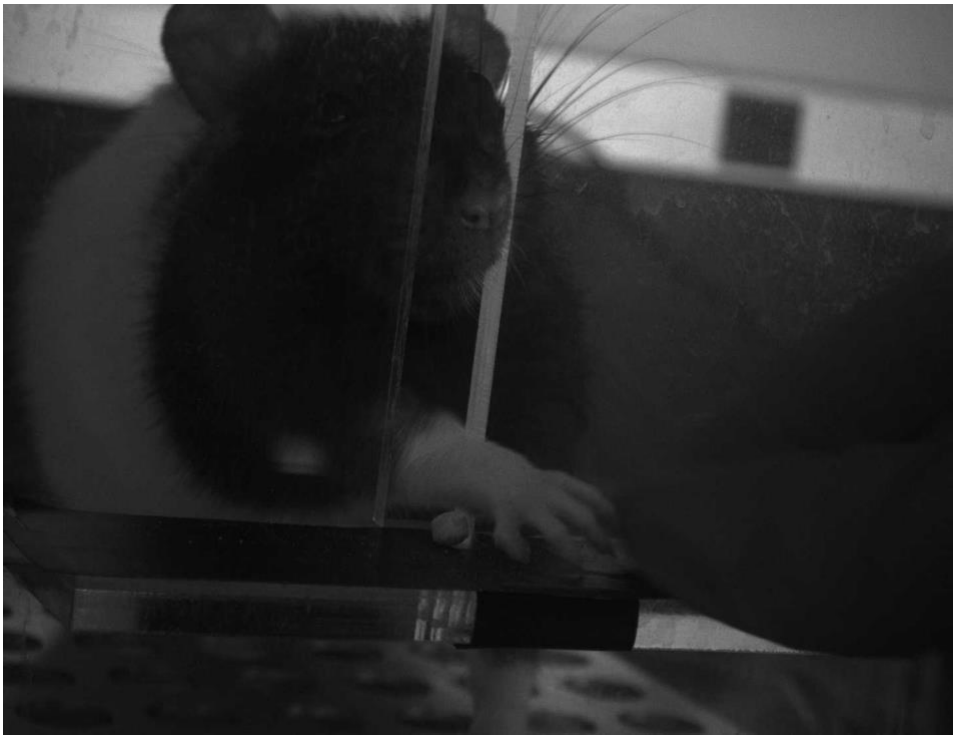

Fig 7. Current camera frame

Challenges

Beginning September 2018, I expected to have the pose-estimation system running preliminary trials by March 2019. I was met with several challenges along the way which stunted my progress leading me unable to achieve these goals.

I ran into many technical difficulties trying to setup DLC. These ranged from installing the necessary packages, many which were incompatible with the latest version of Ubuntu, to understanding how to use the lab's GPUs without disturbing other processes.

In terms of the cameras, the software used to operate them has a steep learning curve and many nuances. To capture one set of images, it takes about six steps on the SpinView software. Thus, getting synchronized capture up and running took far longer than expected.

I also ran into difficulty finding a method to merge the 2D points into 3D ones. Without prior knowledge of camera calibration, it took time to understand which method of calibration would be suitable for the system's needs. The bulk of this work was in understanding how to translate the calibration theory into practice. There were many resources about how the process works mathematically, but I was unable to find much documentation on how to physically find the matrices.

Future Work

There are several tasks which will need to be completed to get the system running at a basic level. In order to be able to merge the two 2D points into 3D points, one will need to find the extrinsic matrices for the camera. This will require printing out another Aruco chessboard and running the extrinsic matrix calibration code, found in the Camera-Calibration repository. Once, the extrinsic matrices are found, they will need to be merged with the intrinsic matrices to find the camera matrix, which is what is ultimately used to project the 2D points.

As of a month ago, DeepLabCut has added 3D calibration functions for multiple cameras to its newest version (2.07) using the same Aruco chessboard setup. It may be easier to use the DLC package to calibrate and save the camera calibrations for use in merging the trajectories.

Another challenge will to be to write code which syncs the real-time system with the frames captured by the cameras. The timestamp feature on the machine is enabled so this is a matter of retrieving the timestamp data and matching it with the clock on the real-time system.

The system also needs to be validated against the current knob turning paradigm. In order to check that the camera system is set up properly, one will need to record a video of the rodent training paradigm. Then it will need to be analyzed with DeepLabCut to get a tagged image and verify the system works by comparing it to behavioral data from the knob. One way to do this is to check if DLC tags the position of the paw as over the knob at the same time the knob records it is being touched.

Finally, the high-speed video will quickly take up storage on the machine. One way the storage can be optimized is by buffering the video and then dumping unnecessary data, similar to how the real-time system optimizes the amount of tetrode recordings saved. Unnecessary data will not include any of the behaviors we are interested in observing. Allowing for 5 seconds of buffering time will allow the program to save the start of an identified behavior (i.e. if the rat is identified as interacting with the knob, the buffering will allow for the beginning of the task, the original reach, to be recorded).

Once the system is complete, it should provide pose-estimation capabilities similar to those seen in the original DLC paper. Figure 8. from the paper, shows how the algorithm should look when applied to a rat completing a grasping task.[7] Once this project is complete, it should provide similar data, however from two different angles.

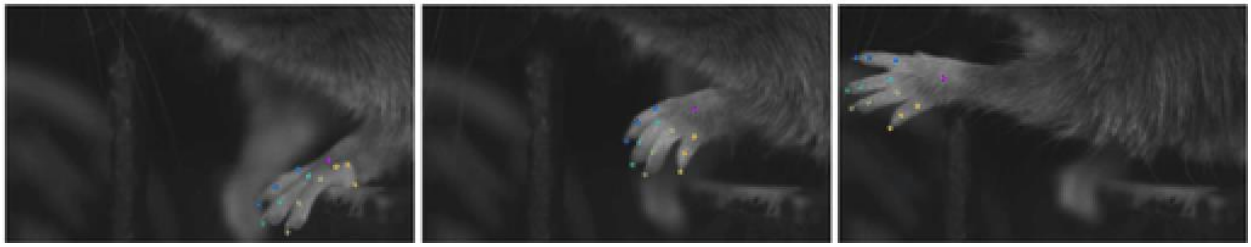

Fig. 8 DeepLabCut applied to a video of a rat completing a grasping task[7]

The new DLC 2.07 documentation shows an example of how the merged trajectories should appear as seen in Figure 9. Each frame shows a different angle of the same image. The third frame then shows the points from each image merged.[15]

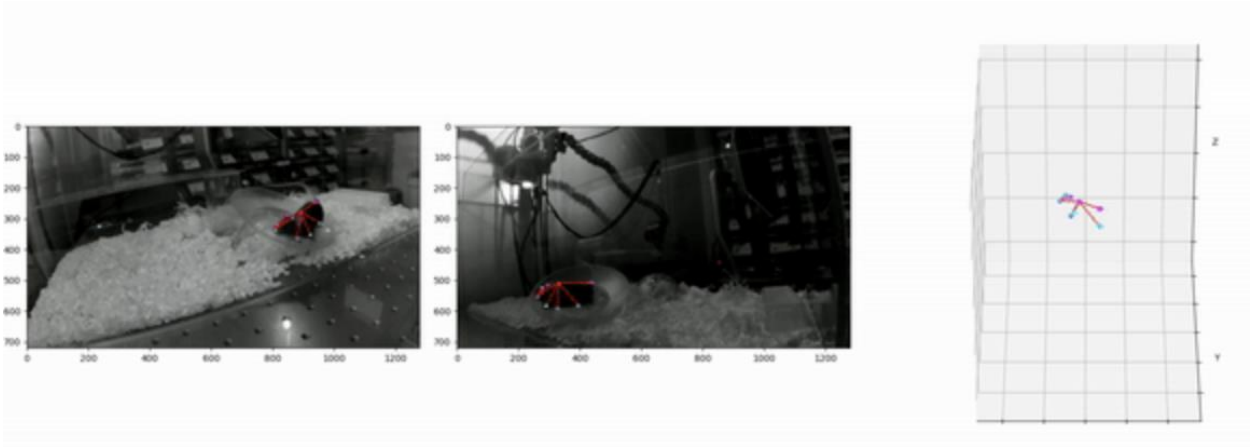

Fig. 9 A 3D representation of rat movement using DLC's new camera calibration functions

References

1. Goldman, B. (2017, February 21). Brain-computer interface advance allows fast, accurate typing by people with paralysis. Retrieved from https://med.stanford.edu/news/allnews/2017/02/brain-computer-interface-allows-fast-accurate-typing-by-people-withparalysis.html

2. Madinabeitia-Mancebo, E., Madrid, A., Jácome, A. *et al.* Temporal dynamics of muscle, spinal and cortical excitability and their association with kinematics during three minutes of maximal-rate finger tapping. *Sci Rep* **10,** 3166 (2020). https://doi.org/10.1038/s41598-020-60043-0

3. Kawai, R., Markman, T., Poddar, R., Ko, R., Fantana, A. L., Dhawale, A. K., Kampff, A. R., & Ölveczky, B. P. (2015). Motor cortex is required for learning but not for executing a motor skill. *Neuron*, *86*(3), 800–812. https://doi.org/10.1016/j.neuron.2015.03.024

4. Hatsopoulos, N. G., & Suminski, A. J. (2011). Sensing with the motor cortex. *Neuron*, *72*(3), 477–487. https://doi.org/10.1016/j.neuron.2011.10.020

5. Bimbi, M., Festante, F., Coudé, G., Vanderwert, R. E., Fox, N. A., & Ferrari, P. F. (2018). Simultaneous scalp recorded EEG and local field potentials from monkey ventral premotor cortex during action observation and execution reveals the contribution of mirror and motor neurons to the mu-rhythm. *NeuroImage*, *175*, 22–31. doi: 10.1016/j.neuroimage.2018.03.037

6. Dhawale, A., Poddar, R., Wolff, S., Normand, V., Kopelowitz, E., & Ölveczky, B. (2017, September 8). Automated long-term recording and analysis of neural ... Retrieved November 28, 2018, from https://pdfs.semanticscholar.org/21fb/e569e158ede50f81de9ecc141ae0c26afb41.pdf

7. Mathis, Alexander, et al. “DeepLabCut: Markerless Pose Estimation of User-Defined Body Parts with Deep Learning.” Nature Neuroscience, vol. 21, no. 9, 2018, pp. 1281–1289., doi:10.1038/ s41593-018-0209-y

8. Maghsoudi, Omid Haji, et al. “Superpixels Based Marker Tracking vs. Hue Thresholding in Rodent Biomechanics Application.” 2017 51st Asilomar Conference on Signals, Systems, and Computers, 2017, doi:10.1109/acssc.2017.8335168.

9. Ellens, Damien J., et al. “An Automated Rat Single Pellet Reaching System with High-Speed Video Capture.” Journal of Neuroscience Methods, vol. 271, 2016, pp. 119–127., doi:10.1016/

10. Cao, Zhe, et al. “Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, doi:10.1109/cvpr.2017.143.

11. Insafutdinov, Eldar, et al. “DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model.” Computer Vision – ECCV 2016 Lecture Notes in Computer Science, Nov. 2016, pp. 34–50., doi:10.1007/978-3-319-46466-4_3.

12. Configuring Synchronized Capture with Multiple Cameras. (n.d.). Retrieved March 26, 2020, from https://www.flir.com/support-center/iis/machine-vision/application-note/configuring-synchronized-capture-with-multiple-cameras

13. Kitani, K. (n.d.). 16-385 Computer Vision. 16-385 Computer Vision. Pittsburg. Retrieved from http://www.cs.cmu.edu/~16385/s17/Slides/11.1_Camera_matrix.pdf

14. (2019, January 28). Retrieved from https://github.com/abhishek098/camera_calibration

15. https://github.com/AlexEMG/DeepLabCut/blob/master/docs/Overviewof3D.md